Near-Lossless Gradient Compression for Data-Parallel Distributed DNN Training

Paper Information

Title: Near-Lossless Gradient Compression for Data-Parallel Distributed DNN Training

Authors: Xue Li, Kun Qian (Alibaba Group), Cheng Guo, Mingwei Xu (Tsinghua University), Menghao Zhang, Mengyu Yang (Unaffiliated)

Conference: SoCC '24, November 20–22, 2024, Redmond, WA, USA

Research Background

1. 生成式 AI 模型的巨大成功极大地推动了分布式深度神经网络（DNN）训练的需求。

2. 随着数据并行（DP）组的数量增加，超大规模模型的训练速度呈次线性增长，这可能导致性能下降高达 70%。

3. 现有的梯度压缩方法依赖于有损或无损压缩。例如：Top-k、Random-k，以及 ZSTD 和 Zlib。

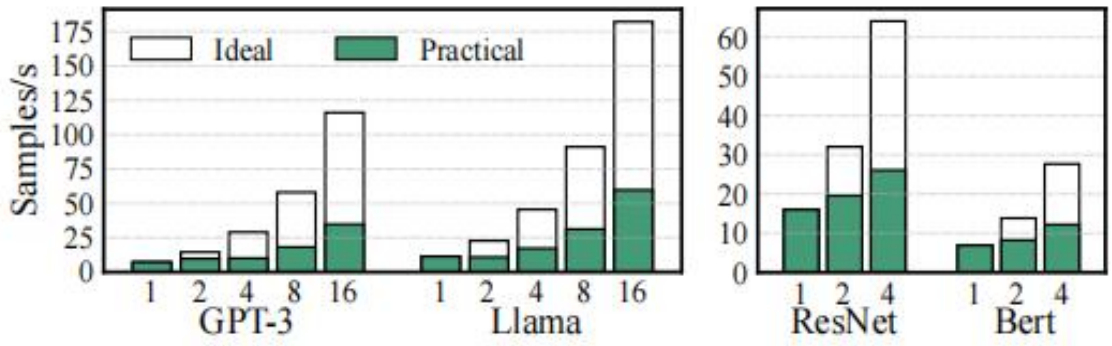


Figure 1: Training speedup of GPT-3, Llama, ResNet and Bert with increasing DP groups (more GPUs used), tested on Alibaba Cloud PAI.

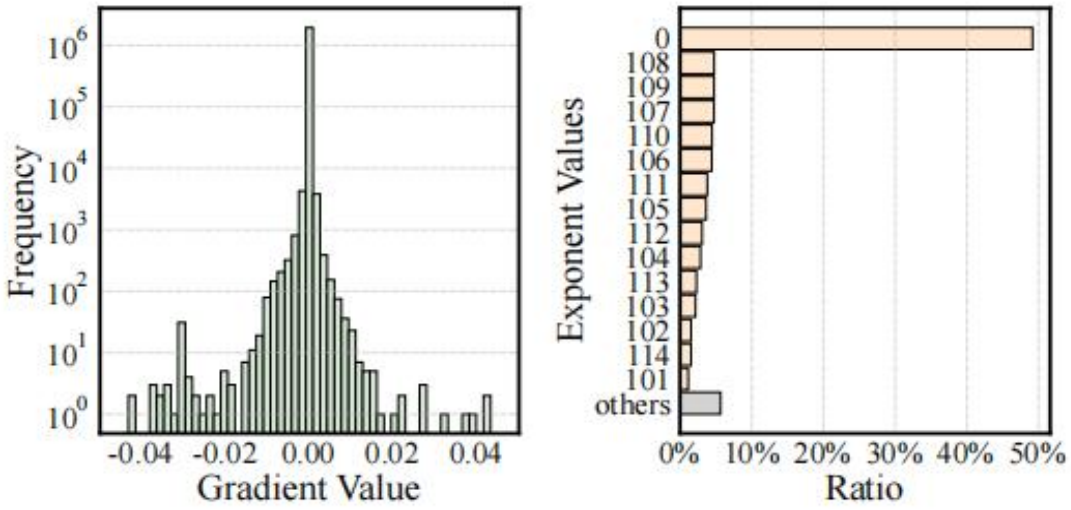
Motivation

要点 1：观察到训练期间梯度值的分布集中在零附近。

1.随着训练的进行，模型逐渐收敛，意味着其参数趋于稳定。

2.梯度的大小逐渐趋向于零。

3.梯度的指数部分逐渐收敛到几个特定值。



(a) Gradient distribution. (b) Exponent distribution.

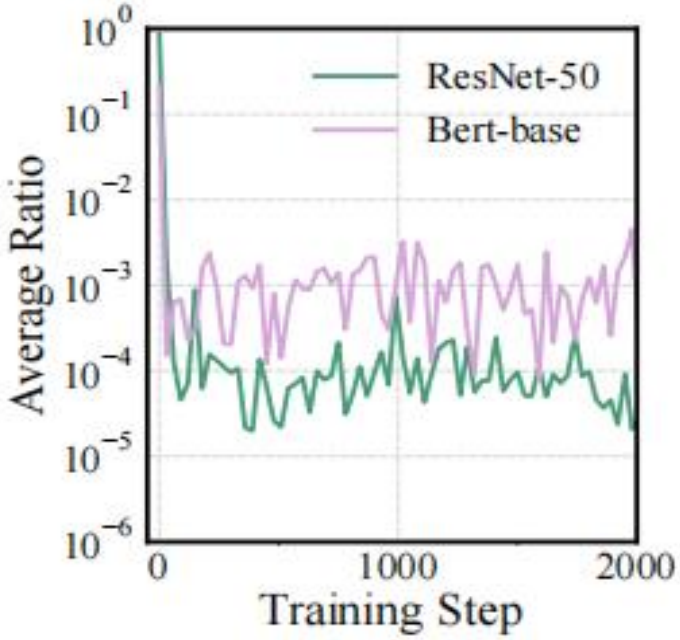
Figure 2: Distribution of gradient values and gradient exponents during the training of a ResNet-50 model using FP32.

要点 2：随着训练的进行，梯度相对于其对应参数的相对大小显著减小。

1.梯度通常比参数小 3 到 5 个数量级。

2.在 DNN 模型中，随着训练的进行，梯度逐渐趋向于更小的值，而参数保持相对一致的幅度。

3.当梯度的幅度显著小于其对应参数的幅度时，梯度尾数中的低位对参数的影响微乎其微。

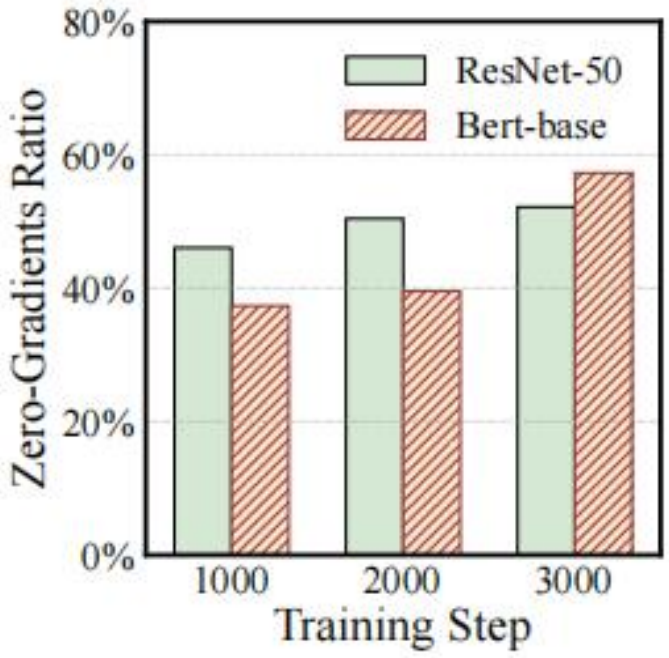


(a) The ratio of gradient to its corresponding parameter.

要点 3：在整个训练过程中，零值梯度普遍存在，并且它们不对模型参数的变化做出贡献。

1.随着训练的推进，一定比例的梯度变为零是常见的。

2.由于这些零梯度不对参数的更新做出贡献，因此它们可以被有效地剪枝。



(b) Proportion of zero-valued gradients during training.

GComp Overview

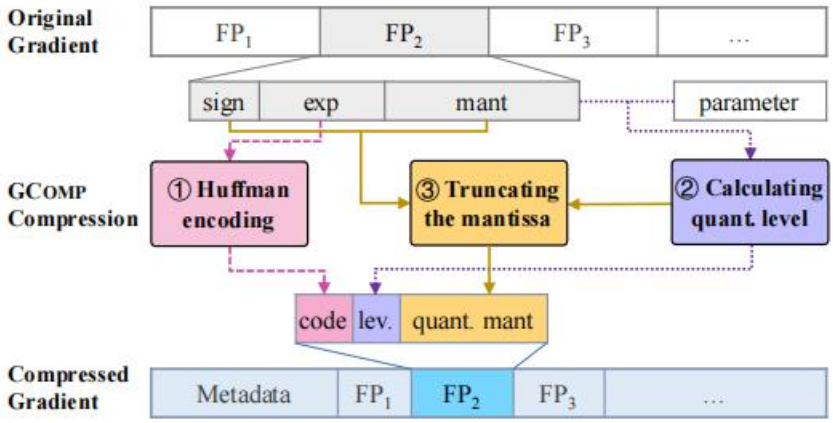


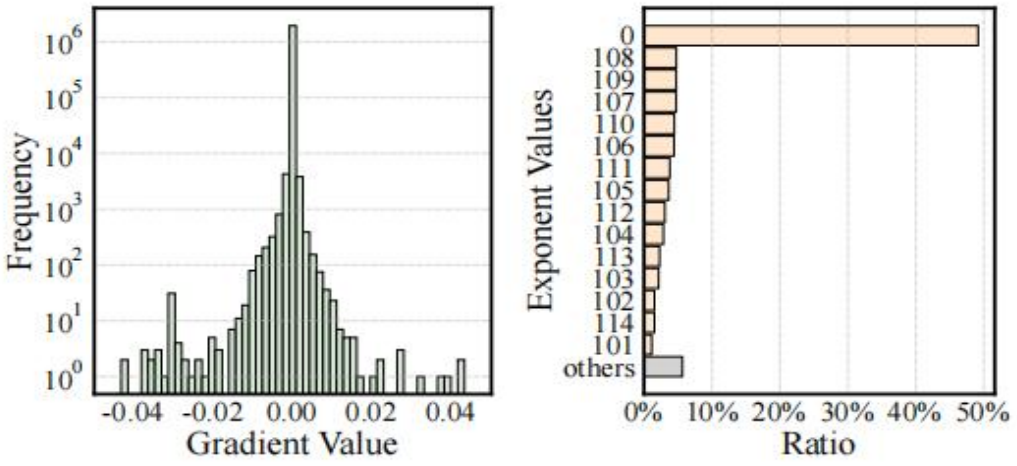
Figure 4: Overall overview of the GCOMP scheme for gradient compression.

Exponent Compression

设计考虑：

1. 可以针对偏斜分布进一步优化哈希表。

2. 在我们的特定上下文中，标准霍夫曼编码仍然具有局限性。



(a) Gradient distribution. **(b) Exponent distribution.**

Figure 2: Distribution of gradient values and gradient exponents during the training of a ResNet-50 model using FP32.

原生霍夫曼编码的局限性

1. 在某些极端情况下，生成的霍夫曼码的长度在最坏情况下（即完全偏斜的树）可能长达（2^8 - 1）位。

2. 随着训练的进行，指数的最优霍夫曼编码可能会发生变化。

3. 霍夫曼解码过程可能效率低下。

Table 1: An example of the native and optimized Huffman encoding, where the 10101111 in bold stands for the special code and the subsequent 8-bit sequence is the raw representation of the exponents.

Source	Native Huffman	Optimized Huffman
0	011	011
1	0011	0011
...	...	...
32	1010111110100101010	**10101111**00100000
...	...	...
138	1101110111001101l0	**10101111**10001010
...	...	...
255	1001	1001

长码的完整传输

1.在编码过程中，如果生成的码的长度超过预定阈值，则它将被替换为特殊码，以指示应原样传输原始指数值。

2.假设源值的位长度为 n_s，码长度的阈值为 n_θ，则所有长度超过 n_θ的码都将被替换为特殊码。

$$n_f = n_\theta + n_s.$$

霍夫曼编码表的动态调整

1.在 GComp 中，霍夫曼编码表是利用训练开始前关于指数值分布的先验知识构建的。

2.为了适应分布的变化，此表会定期更新，例如，每 50 次迭代后更新一次。

3.更新霍夫曼编码表需要额外的网络带宽。为了消除这一副作用，我们采用异步交互来更新此表。

基于查找表的解码

1.霍夫曼解码过程通常涉及从编码段中顺序读取位，并从根到叶遍历编码树。

2.我们的优化编码算法对码长度施加上限。

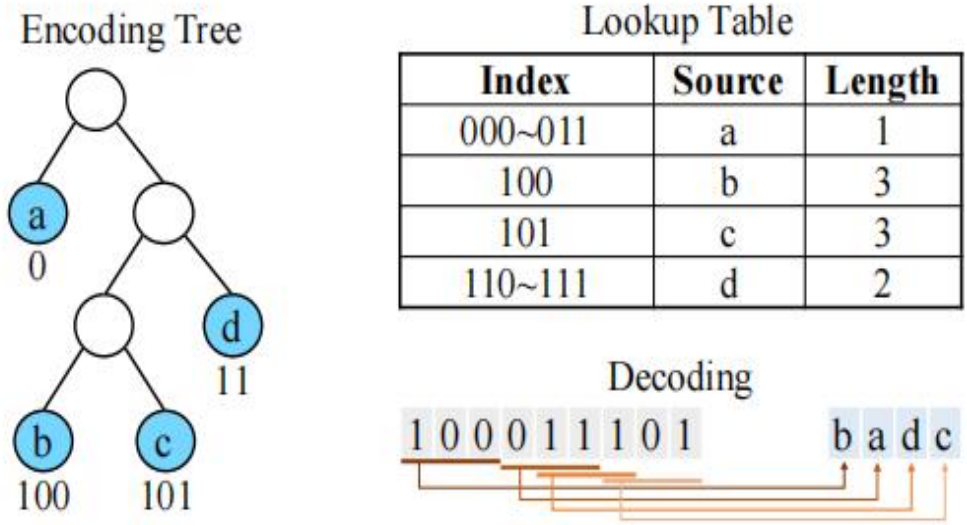


Figure 5: Encoding tree and lookup table with $n_\theta = 3$.

MANTISSA COMPRESSION

多级量化

1.引入多级量化策略来优化尾数压缩，根据梯度相对于其对应参数幅度在优化器中的相对重要性来定制压缩级别。

2.GComp 将量化分为四个不同级别，每个级别由特定数量的截断位表示，并在编码段中用 2 位二进制码表示。

3.这四个级别对应于不同的截断位数，例如，在 FP32 格式的 23 位尾数中分别截断 0、6、12 和 18 位。

Table 2: The quantization metrics for different optimizers.

Optimizer	Updating Method	Quantization Metric δ
SGD	$g_t = g_t + \lambda\theta_{t-1}$ ①; $\theta_t = \theta_{t-1} - \eta g_t$ ②.	$\frac{\theta_{t-1}}{\eta g_t} - \frac{\lambda\theta_{t-1}}{g_t}$
Momentum	$g_t = g_t + \lambda\theta_{t-1}$ ①; $b_t = \mu b_{t-1} + (1-\tau)g_t$ ②; $\theta_t = \theta_{t-1} - \eta b_t$ ③.	$\frac{\theta_{t-1}-\eta\mu b_{t-1}}{\eta(1-\tau)g_t} - \frac{\lambda\theta_{t-1}}{g_t}$
Nesterov Momentum	$g_t = g_t + \lambda\theta_{t-1}$ ①; $b_t = \mu b_{t-1} + g_t$ ②; $g_t = g_t + \mu b_t$ ③; $\theta_t = \theta_{t-1} - \eta g_t$ ④.	$\frac{(1-\eta\lambda(1+\mu))\theta_{t-1}-\eta\mu^2 b_{t-1}}{\eta(1+\mu)g_t}$
AdaGrad	$\tilde{\eta} = \eta/(1+(t-1)\lambda_{lr})$ ①; $g_t = g_t + \lambda\theta_{t-1}$ ②; $r_t = r_{t-1} + g_t^2$ ③; $\theta_t = \theta_{t-1} - \frac{\tilde{\eta} g_t}{\sqrt{r_t}+\epsilon}$ ④.	$\frac{\theta_{t-1}(\sqrt{r_t}+\epsilon)}{\tilde{\eta} g_t} - \frac{\lambda\theta_{t-1}}{g_t}$
RMSProp	$g_t = g_t + \lambda\theta_{t-1}$ ①; $v_t = \alpha v_{t-1} + (1-\alpha)g_t^2$ ②; $\theta_t = \theta_{t-1} - \frac{\eta g_t}{\sqrt{v_t}+\epsilon}$ ③.	$\frac{\theta_{t-1}(\sqrt{v_t}+\epsilon)}{\eta g_t} - \frac{\lambda\theta_{t-1}}{g_t}$
Adam	$g_t = g_t + \lambda\theta_{t-1}$ ①; $m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$ ②; $v_t = \beta_2 v_t + (1-\beta_2)g_t^2$ ③; $\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$ ④; $\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$ ⑤; $\theta_t = \theta_{t-1} - \frac{\eta\widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon}$ ⑥.	$\frac{\theta_{t-1}(\sqrt{\widehat{v_t}}+\epsilon)(1-\beta_1^t)-\eta\beta_1 m_{t-1}}{\eta(1-\beta_1)g_t} - \frac{\lambda\theta_{t-1}}{g_t}$
AdamW	$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$ ①; $v_t = \beta_2 v_t + (1-\beta_2)g_t^2$ ②; $\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$ ③; $\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$ ④; $\theta_t = \theta_{t-1} - \eta(\frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon} + \lambda\theta_{t-1})$ ⑤.	$\frac{\theta_{t-1}(\sqrt{\widehat{v_t}}+\epsilon)(1-\beta_1^t)(1-\eta\lambda)-\eta\beta_1 m_{t-1}}{\eta(1-\beta_1)g_t}$

零值梯度的剪枝

1. 鉴于零值浮点数的指数和尾数均为零，尾数可以安全地丢弃。

2. GComp 仅保留用于表示零值梯度的指数部分的霍夫曼编码，因为零值浮点数的指数和尾数均为零。

3. 在 DNN 训练期间，GComp 将所有 FP32 格式的非规格化数视为零，因为它们很少出现，并且从零梯度剪枝产生的误差可忽略不计。

IMPLEMENTATION

压缩/解压缩过程

单个值的编码

1. 指数编码

2. 量化级别计算

3. 尾数量化和截断

单个值的解码

1. 访问压缩数据的霍夫曼编码部分，并使用查找表检索原始指数值。

2. 读取量化级别和量化尾数段。

3. 重建尾数和符号位。

数据布局

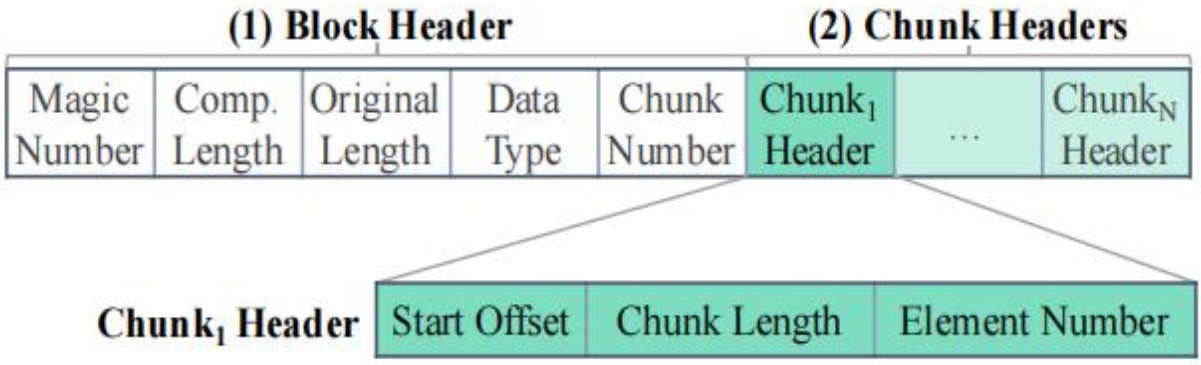


Figure 7: Metadata layout for a compressed data block.

与 AllReduce 的集成

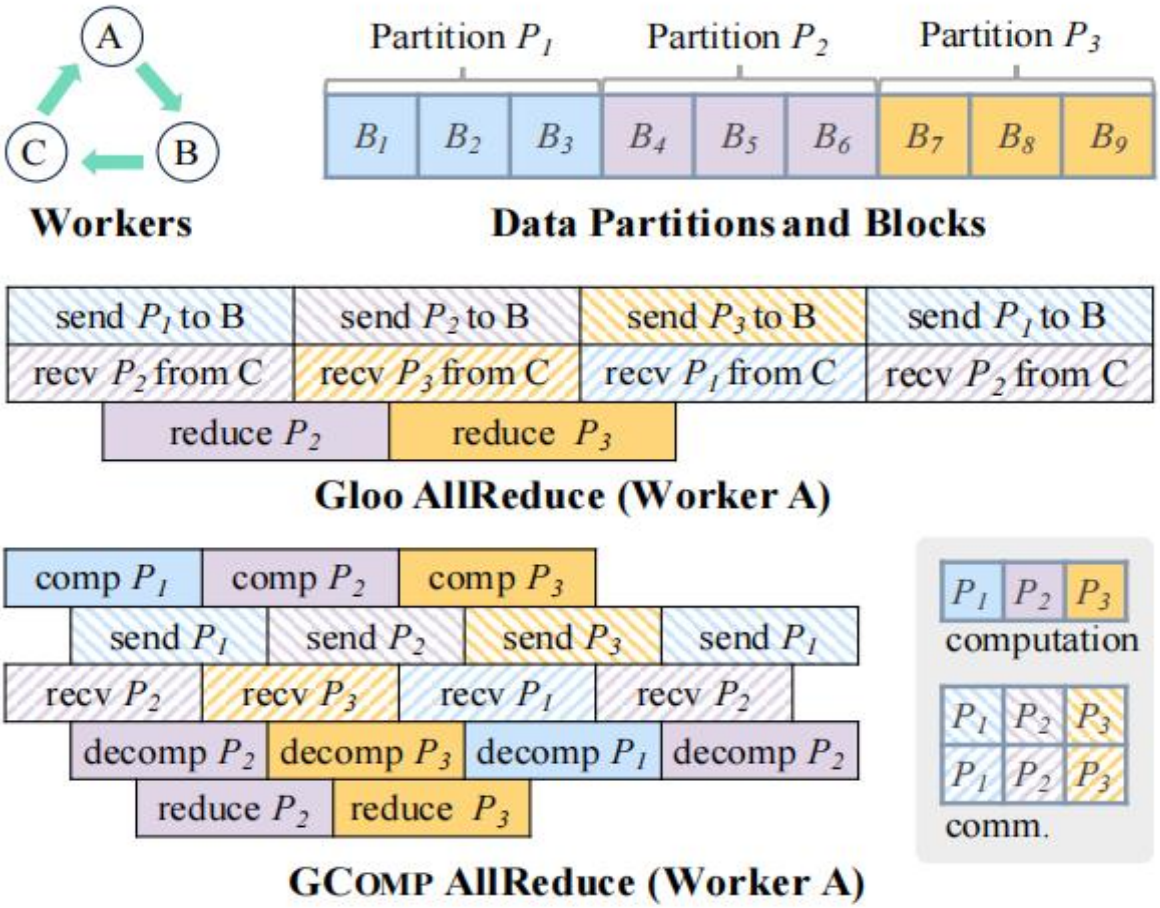


Figure 8: Integration of GCOMP into the AllReduce operation (with three workers).

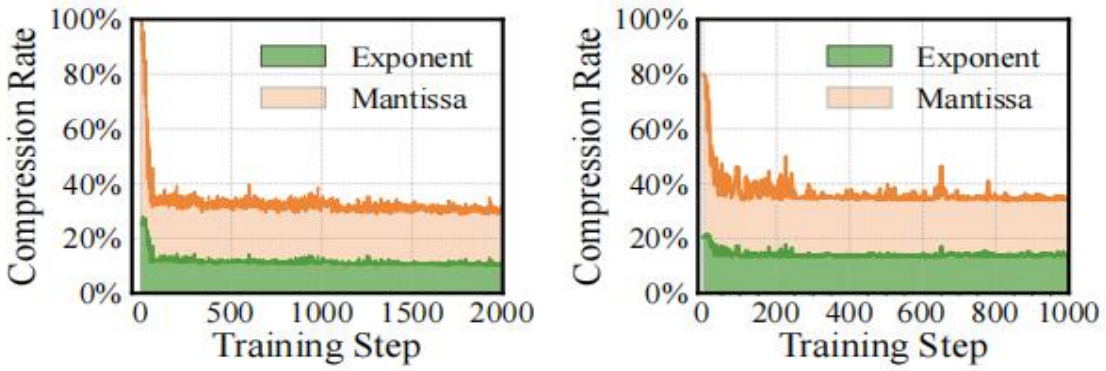
Experimental Setup

平台：所有实验均在阿里云提供的机器学习平台 PAI 上进行。为评估目的，共使用了 4 个实例。每个实例配备单个 NVIDIA A100（40GB）GPU、12 个 CPU 核心和 94GB CPU 内存。每个实例在分布式训练设置中作为单独的数据并行组，实例之间通过 10Gbps 以太网网卡连接。

工作负载：在 ImageNet1K 数据集上训练包含 2360 万个参数的 ResNet-50，以及在 SST 数据集上训练包含 1.1 亿参数的 Bert-base。

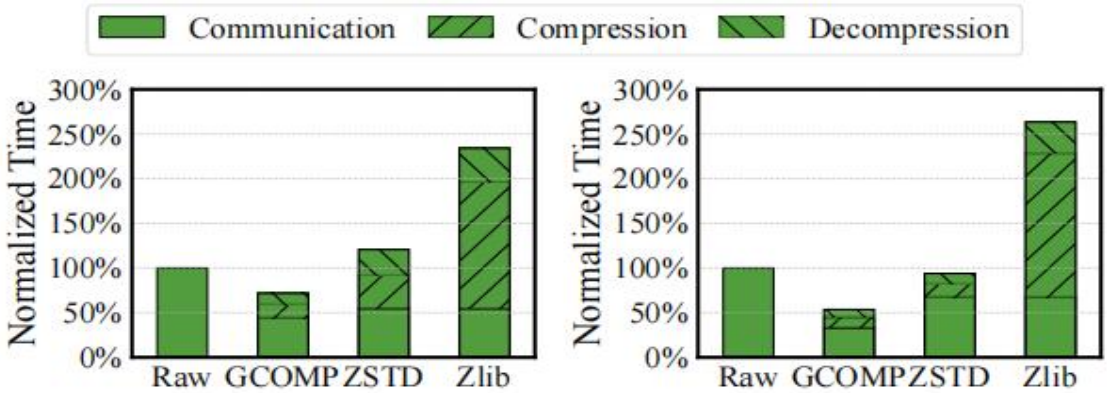
框架：我们实现的 GComp 将压缩和解压缩模块集成到 Gloo 库的 AllReduce 中。

Compression Efficiency



(a) ResNet-50. (b) Bert-base.

Figure 9: Compression rate changes during training.



(a) ResNet-50. (b) Bert-base.

Figure 10: The normalized time for the AllReduce with different compression methods.

Exponent Compression Benchmarks

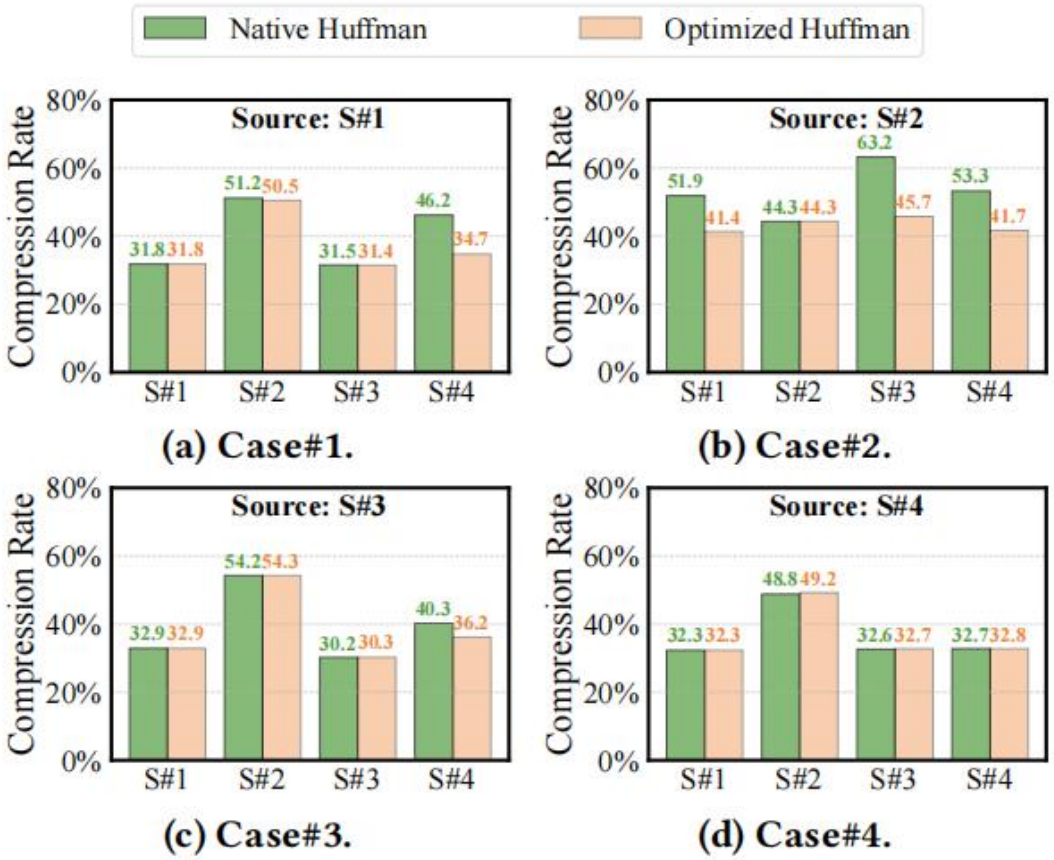


Figure 12: Compression rate of the exponent part when using the optimized Huffman code and the native Huffman code, each subfigure using a different information source for building the Huffman codebook.

Mantissa Compression Benchmarks

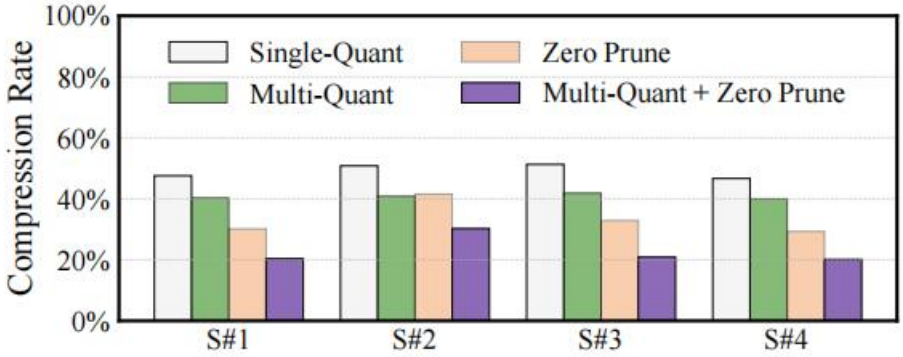
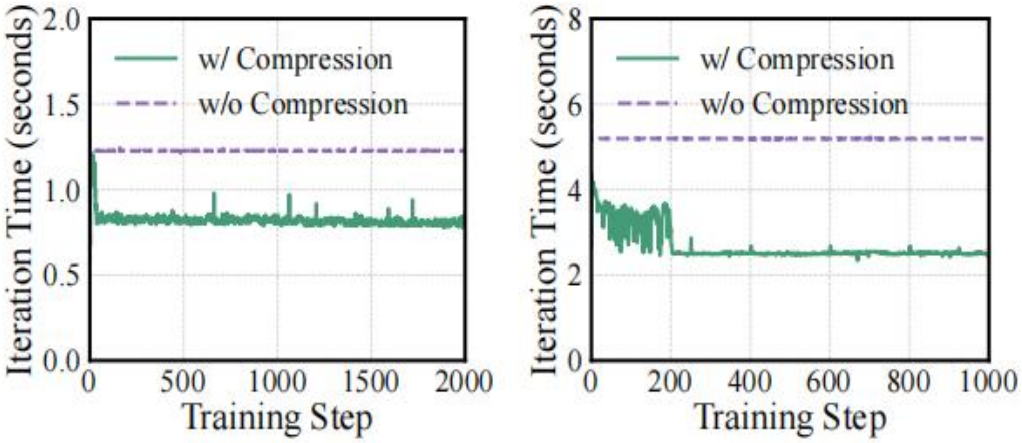


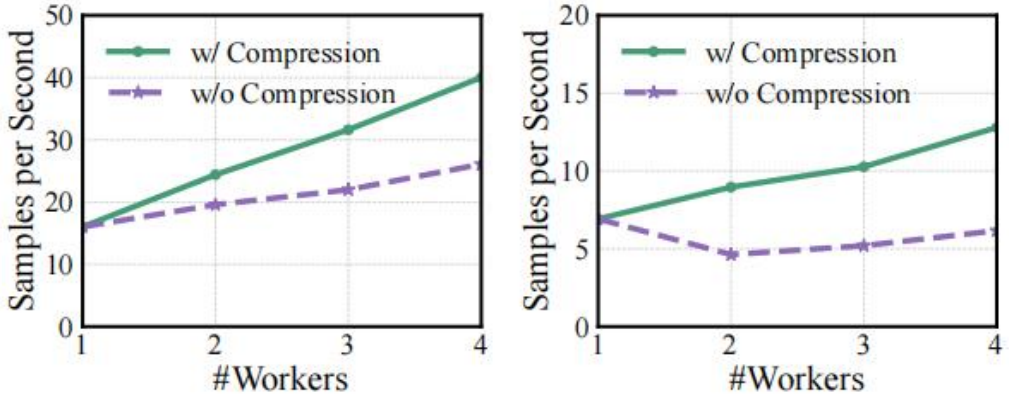
Figure 14: Compression rate of the mantissa part using different strategies.

Training Efficiency



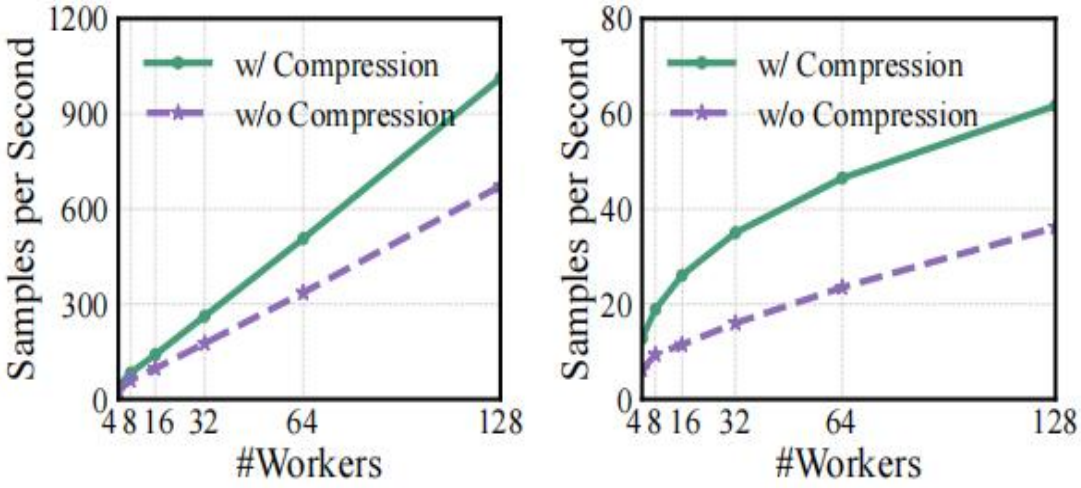
(a) ResNet-50. **(b) Bert-base.**

Figure 15: Iteration time with and without GComp.



(a) ResNet-50. **(b) Bert-base.**

Figure 16: Training scalability with and without GComp (evaluation results).



(a) ResNet-50. (b) Bert-base.

Figure 17: Training scalability with and without GCOMP (simulation results).

Impact on Model Convergence

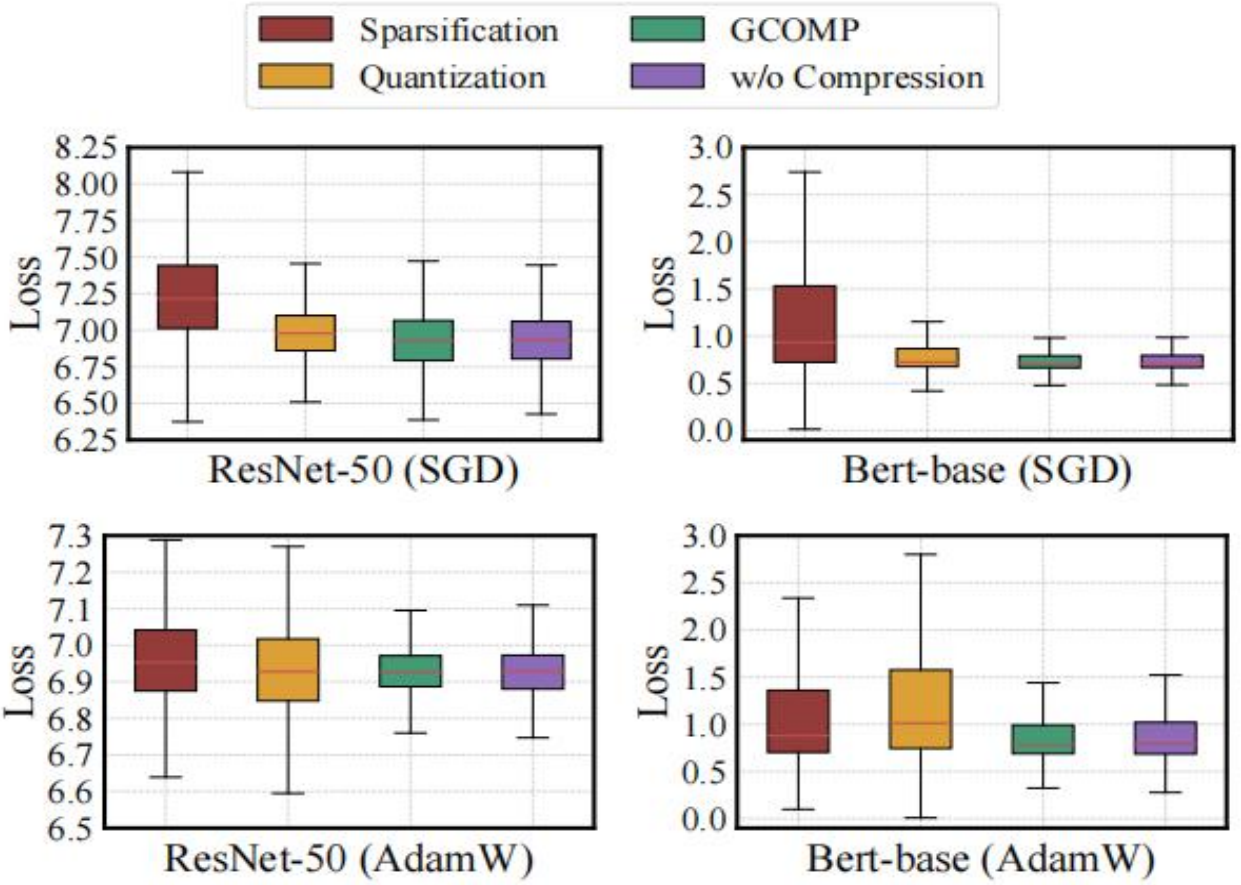


Figure 18: Distribution of loss deviations compared with the baseline training without any compression.

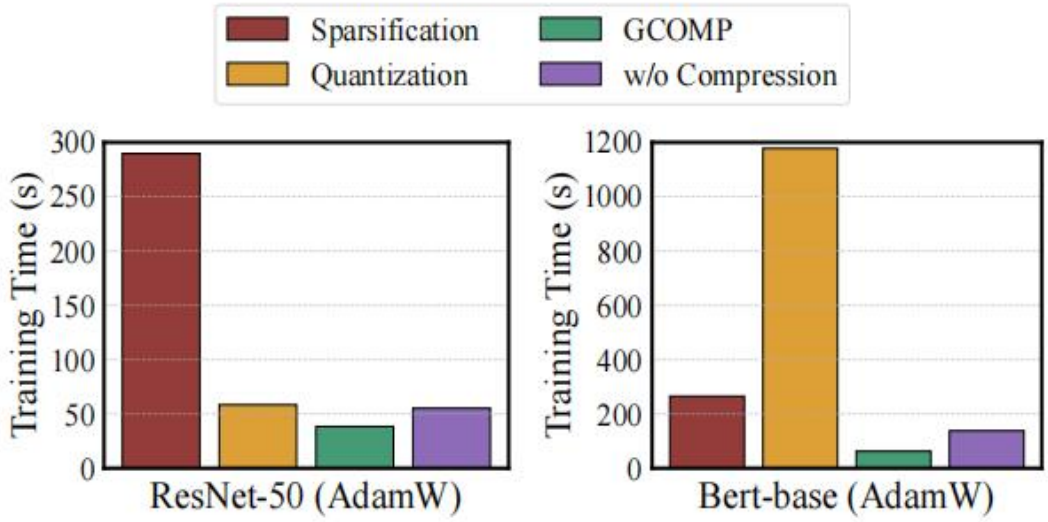


Figure 20: End-to-end training time to reach specific convergence levels.

Conclusion

1.GComp 是一种近无损梯度压缩方案，旨在解决数据并行分布式 DNN 训练中的通信开销问题。

2.通过利用梯度分布的统计特性，GComp 实现了近无损压缩，有效减少了通信量高达 67.1%。

3.GComp 将训练效率提高了多达 1.9 倍，证明了其作为加速和扩展 DNN 训练的潜在解决方案的前景。